CS734 Assignment 2: Algebra, Logic & Software Design & Verification

Due: 1330 Thursday March 21, 2002

Download from the course website the file called `a2.dmp`. Put this file in the directory where you wish to do your work and undump it with the *PVS—Files and Theories—undump-pvs-files* command. This will create several files. All of your PVS work for this assignment should be done in the `a2.pvs` file at the locations indicated by the comments in the file. Email your completed assignments to lawford@groke.mcmaster.ca with the subject "Assignment2".

1. Equivalence Kernels and Software Verification (15 marks)

In the following, let V_1, V_2 and V_3 be nonempty sets. Any function $f : V_1 \rightarrow V_3$ induces an equivalence relation $\ker(f)$, the *equivalence kernel of* f, given by

$$(v_1, v_1') \in \ker(f) \text{ if and only if } f(v_1) = f(v_1')$$

where $(v_1, v_1') \in V_1 \times V_1$.

The PVS definition of $\ker(f)$ appearing in theory `equivker` make use of the `relations` theory from the prelude file.

We can define a partial order on equivalence relations as follows: Let E_1 and E_2 be equivalence relations on V_1. Then we say that E_1 is a refinement of E_2, written $E_1 \leq E_2$ iff $\forall v_1, v_1' \in V_1 : (v_1, v_1') \in E_1 \rightarrow (v_1, v_1') \in E_2$.

Consider the following result from discrete mathematics:

Theorem: *Given two functions with the same domain, $f : V_1 \rightarrow V_3$ and $g : V_1 \rightarrow V_2$, then there exists $h : V_2 \rightarrow V_3$ such that the diagram in Figure 1 commutes iff* $\ker(g) \leq \ker(f)$.

$$\begin{tikzcd} V_1 \arrow[rr, "f"] \arrow[dr, "g"'] & & V_3 \\ & V_2 \arrow[ur, dashed, "h"'] & \end{tikzcd}$$

Figure 1: Commutative diagram for $(\exists h : V_2 \rightarrow V_3) h \circ g = f$ iff $\ker(g) \leq \ker(f)$

The interpretation of this result is that for h to exist, g must retain as much or more information about its domain than f.

Given $f : V_1 \rightarrow V_3$ we define the *image* of f, denoted $\text{Im}(f)$, as follows:

$$\text{Im}(f) := \{v_3 \in V_3 | \exists v_1 \in V_1 (f(v_1) = v_3)\}$$

Let us now consider the dual situation shown in Figure 2 where we are given f and h and want to know if there exists g such that $h \circ g = f$.

Theorem: *Given two functions with the same codomain, $f : V_1 \rightarrow V_3$ and $h : V_2 \rightarrow V_3$, then there exists $g : V_1 \rightarrow V_2$, such that the diagram in Figure 2 commutes iff $Im(f) \subseteq Im(h)$.*

The interpretation of this result is that for g to exist, h must be able to reach every point that f can reach.

These two theorems have already been stated and proved in the PVS file in the theory . You will now use them in PVS to prove some properties of the commutative diagrams for the Systematic Design Verification procedure.

$$
\begin{tikzcd}
V_1 \arrow[rr, "f"] \arrow[dr, dashed, "g"'] & & V_3 \\
& V_2 \arrow[ur, "h"'] &
\end{tikzcd}
$$

Figure 2: Commutative diagram for $\exists g : V_1 \to V_2 (h \circ g = f)$ iff $\mathrm{Im}(f) \subseteq \mathrm{Im}(h)$

a) Given a software requirements function $REQ : M \to C$ together with abstraction functions $Abst_M : M \to M_p$ and $Abst_C : C \to C_p$, state and prove in PVS necessary and sufficient conditions for the existence of an implementation function $SOF_{REQ} : M_p \to C_p$ such that:

$$SOF_{REQ} \circ Abst_M = Abst_C \circ REQ$$

$$
\begin{tikzcd}
M \arrow[r, "REQ"] \arrow[d, "Abst_M"'] & C \arrow[d, "Abst_C"] \\
M_p \arrow[r, dashed, "SOF_{REQ}"] & C_p
\end{tikzcd}
$$

b) Given a software requirements function $REQ : M \to C$ and functions $Abst_M : M \to M_p$ and $Abst_{C_p} : C_p \to C$. State and prove necessary and sufficient conditions for the existence of an implementation function $SOF : M_p \to C_p$ such that:

$$Abst_{C_p} \circ SOF \circ Abst_M = REQ$$

$$
\begin{tikzcd}
M \arrow[r, "REQ"] \arrow[d, "Abst_M"'] & C \\
M_p \arrow[r, dashed, "SOF"] & C_p \arrow[u, "Abst_{C_p}"']
\end{tikzcd}
$$

c) In this problem we revisit a version of the verification of a simplified pressure sensor trip from the lecture slides. The proposed specification and the actual implementation for the sensor trip are give in Figure 3 by `f_PressTrip` and `PTRIP`, respectively.

Theorem `Sentrip1` is the block comparison theorem to verify that the implementation satisfies the specification. This theorem is unprovable. In fact, it is currently impossible to change the definition of `PTRIP` so that it will satisfy the specification `f_PressTrip`. Using PVS state and prove a theorem to this effect.

sentrip : THEORY
BEGIN

Trip : TYPE = {Tripped, NotTripped}

AItype : TYPE = $\{\mathrm{i} : \mathrm{nat} \mid 0 \leq \mathrm{i} \land \mathrm{i} \leq 5000\}$

f_PressTrip(Pressure : real, f_PressTripS1 : Trip) : Trip = TABLE

Pressure $\leq$ 2400	2400 $<$ Pressure $\land$ Pressure $<$ 2450	Pressure $\geq$ 2450
NotTripped	f_PressTripS1	Tripped

ENDTABLE

PTRIP(PRES : AItype, PREV : bool) : bool = TABLE

PRES $\leq$ 2400	2400 $<$ PRES $\land$ PRES $<$ 2450	PRES $\geq$ 2450
FALSE	PREV	TRUE

ENDTABLE

Trip2bool(TripVal : Trip) : bool = TABLE

TripVal = Tripped	TripVal = NotTripped
TRUE	FALSE

ENDTABLE

bool2Trip(BoolVal : bool) : Trip = TABLE

BoolVal = TRUE	BoolVal = FALSE
Tripped	NotTripped

ENDTABLE

real2AItype(x : real) : AItype = TABLE

$x \leq 0$	$0 < x \land x < 5000$	$x \geq 5000$
0	floor(x)	5000

ENDTABLE

Sentrip1 : THEOREM
($\forall$ (Pressure : real, f_PressTripS1 : Trip) :
f_PressTrip(Pressure, f_PressTripS1) =
bool2Trip(PTRIP(real2AItype(Pressure), Trip2bool(f_PressTripS1))))

END sentrip

Figure 3: Formatted PVS specification for pressure sensor trip example

2. When you undumped the a2.dmp file it created the additional files `Clocks.pvs`, `Held_For.pvs` and `TimerGeneral.pvs`. Load each of these files in the above order and as each file is loaded run the PVS command *PVS—Prover Invocation—prove-pvs-file*. All of the TCCs and theorems should prove (note: this is the PVS 2.3 version of the files PVS 2.4 chokes on some of the proofs). PVS is now aware of these results and you can use any of the results from these files in your work. Now switch back to your `a2.pvs` file and answer the following questions.

a) In the `modular_SenLock` theory, create a new definition of `ELOCK` that implements the Software Requirements Specification (SRS) function `SenLock` with a three valued output as was done in the slides, only this time make use of the `TimerUpdate` function from the `TimerGeneral.pvs` file to update the timer `lLockDly`.

b) Show that your implementation is correct by proving the block comparison theorem `SensorLock_Block`. (Hint: Use a lemma to show that `lLockDly` is updated in a similar fashion to `Timer` and then make use of the main result of the `TimerGeneral.pvs` file to show that your implementation correctly implements the `Held_For`.)